EFFICIENT APPROACHES FOR THE SOLUTION OF SOME HARD CONSTRAINED OPTIMIZATION PROBLEMS IN GRAPHS

A

Synopsis submitted in the partial fulfillment for the requirement of

Doctor of Philosophy

In

Physics and Computer Science

Submitted by

Neha

Under the Supervision of

Dr. V. Prem Prakash

Co-supervisor
Department of Electrical Engineering
Faculty of Engineering

Prof. C. Vasantha Lakshmi

Supervisor
Department of Physics & Computer Science
Faculty of Science

DEPARTMENT OF PHYSICS AND COMPUTER SCIENCE

FACULTY OF SCIENCE

DAYALBAGH EDUCATIONAL INSTITUTE (DEEMED UNIVERSITY)

DAYALBAGH, AGRA – 282005

2021

Contents

1. Introduction 1
2. Constrained Optimization Problems in Graphs 2
   2.1. Degree-Constrained Minimum Spanning Tree (DCMST) Problem 3
      2.1.1. Multi-Period Degree-Constrained Minimum Spanning Tree (MPDCMST) 5
      2.1.2. Bounded-Degree Spanning Tree (BDST) 6
      2.1.3. Minimum-Degree Constrained Minimum Spanning Tree (md-CMST) 7
   2.2. Bounded-Diameter Minimum Spanning Tree (BDMST) Problem 8
      2.2.1. bi-Objective Minimum Diameter-Cost Spanning Tree (bi-MDCST) Problem 10
   2.3. Minimum Label Spanning Tree 11
   2.4. Leaf-Constrained Minimum Spanning Tree Problem 12
   2.5. Optimum Communication Spanning Tree Problem 14
3. Parallelization Approach 15
   3.1. MPI/Open MPI 18
   3.2. CUDA (Compute Unified Device Architecture) 18
4. Meta-heuristics 19
5. Proposed Work 20
6. References 21

1. Introduction

Optimization is maximizing or minimizing some function relative to some set of parameters. It deals with finding, for a given problem, the best of all feasible solutions. The problem is said to be a *discrete optimization problem* if the set of feasible solutions is finite. A combinatorial optimization problem (COP) is one in which the goal is to find (one or more) optimal solutions that optimize a given objective function in a well-defined discrete problem space: a space comprising of a finite set of solutions.

Formally, given a set S of base elements, a cost function C: $S \rightarrow \mathbb{R}$ assigning a cost to each element of S and the set $X \subseteq 2^S$ of feasible subsets of S, we define a COP in general as the problem of finding a minimum (maximum) cost feasible subset as follows.

$$\min_{x \in X} (\max) \sum_{i \in x} C_i$$

Combinatorial optimization problems occur in many areas of engineering and management, for example, the design and analysis of data networks, bridges, waterways; logistics for transport, facility location, telecommunication network design, scheduling, routing, assignment, finance, manufacturing, and so on.

Although the number of solutions to a COP is finite, they are typically exponential in terms of the input size. Most COPs are known to be NP-hard, which means that there is no known polynomial-time algorithm for solving them. Thus, it is not possible to guarantee that an exact solution to the problem can be found within an acceptable time frame. The main focus in combinatorial optimization is therefore on the development of algorithms that provide a satisficing (not necessarily optimal) solution in a reasonable time.

Exact algorithms for a COP always find optimal solutions on all instances of the problem within a bounded time; however, the high complexity of many combinatorial problems prohibits finding an exact solution in a reasonable time. Even when there is a polynomial algorithm for a problem, the degree of the polynomial is typically so large as to require too much time to solve realistic instances (these are also known as long-term problems in the complexity class P). Clearly, in such situations, making use of exact algorithms to reach optimality may be impractical for all but very small problem instances,

whereas approximate algorithms that concentrate on obtaining well feasible (but not necessarily optimal) solutions in much lesser time could be put to good use.

A significant development in approximate algorithms for COPs is the emergence of a class of algorithms called *meta-heuristics*. Meta-heuristics incorporate various heuristic approaches into higher-level frameworks that work with populations of (one or more) candidate solutions and try to improve these solutions by efficiently exploring the search space. Several meta-heuristic strategies like Memetic Algorithm, Grey Wolf optimization, Simulated Annealing (SA), Evolutionary Algorithms (EAs), and Artificial Bee Colony (ABC) Optimization, to name a few, have been developed and applied successfully to a variety of optimization problems.

The *No Free Lunch* theorem [1]states that the performance of all optimization algorithms, amortized over the set of all possible functions, is equivalent. This means that if an optimization algorithm is designed to exploit the particular characteristics of a particular problem or class of COPs, then it could perform well on COPs of that type, but possibly fail with other problem classes. This implies that it is not possible to have a single general-purpose universal optimization strategy; the only way one strategy works better than another is when it is specially adapted to problems under consideration [2]. The key is to make this set of problems for which the strategy works better as large as possible.

2. Constrained Optimization Problems in Graphs

Combinatorial optimization problems on graphs that involve finding a subset of edges of some graph G such that when those edges are removed from G, the graph remains connected, and the sum of graph edges is minimized, have a wide range of applications. These range from physical system design (for example, designing a pipeline network to connect a number of towns while ensuring the smallest possible total pipeline length) to cluster analysis (partitioning a given set of data points into natural groups, or clusters such that data within a cluster are more closely related to one another than to data outside the cluster). Spanning trees are very useful for modeling such COPs. Basically, given a connected, undirected graph G, a spanning tree on G is a connected, acyclic sub-graph that contains all the vertices of G. A minimum spanning tree is a spanning tree, the sum of whose edge weights is minimal. Well-known algorithms such as Boruvka‘s algorithm, Prims’s algorithm and Kruskal’s

algorithm are known to compute minimum spanning trees in polynomial time. However, constraints on the spanning trees often render the search for the lowest cost/weight tree much harder. Putting a bound on, for instance, the degree, diameter, or the number of leaves in the desired minimum spanning tree renders the problem NP-hard, giving rise to the Degree-, Diameter- and Leaf- constrained Minimum Spanning Tree problems respectively. Some hard combinatorial optimization problems in graphs are discussed as follows, and a tabular summary of their applications is provided in

Table 1.

2.1. Degree-Constrained Minimum Spanning Tree (DCMST) Problem

Definition. Given a connected, weighted, undirected graph $G \equiv (V, E)$, find a spanning tree of minimum weight with the additional constraint that no vertex $v \in V$ has a degree more than a given bound B. The degree of a vertex is defined as the number of edges connecting vertex v to any other vertex $w \in V$ in the graph.

The Degree-Constrained Minimum Spanning Tree (DCMST) Problem is NP-hard [3], hence exact algorithms can be applied only for small-sized problem instances. For working with much larger graphs that often arise in real-world applications, several meta-heuristic techniques have been presented in the literature.

Zhou and Gen [4] give a genetic algorithm that uses Prüfer coding to represent chromosomes. Raidl and Julstrom [5] use a weighted coding technique that represents candidate solutions as strings of weights associated with the graph vertices. A Variable Neighbourhood Search (VNS) algorithm for the DCMST problem is given by P. Martins and M.C. deSouza [6] [7].

Two Ant Colony Optimization-based algorithms for the problem are given by Bau, Ho, and Ewe in [8], one of which (called p-ACO in the paper) is motivated by Prim's algorithm and uses graph vertices as solution components. The other algorithm (called k-ACO) uses graph edges as solution components and is based on Kruskal's algorithm. An improved ACO-based algorithm called AB-DCST, given by Bui, Deng, and Zrncic [9] augments the ACO with two local improvement algorithms that try to replace MST edges with lower cost edges.

A hybrid heuristic [10] called CoLaPSO combines the Wedelin Lagrangian heuristic [11] with a particle swarm optimization (PSO) algorithm. The heuristic uses problem representation that simultaneously works in the dual space (Lagrangian multipliers) and the primal space in the form of cost perturbations. CoLaPSO combines the ability to calculate the lower bounds on the objective from the Lagrangian method with the inherent parallelism of the PSO by combining PSO with a well-known Lagrangian heuristic and is shown to perform better than several extant heuristics on standard test problems comprising of Euclidean graphs of up to 200 vertices and non-Euclidean graphs of up to 1000 vertices.

J.A. Torkestani [12] gives an iterative learning automata-based heuristic called LACT that finds a near-optimal solution to the DCMST problem. Vertices reward low weight incident edges and thus learn to connect locally to the degree constrained spanning tree (DCST) through the minimum weight edge subject to the degree constraint.

The performance of the heuristic is compared with a recent best performing meta-heuristic (AB-DCST) on several Euclidean and Non-Euclidean graphs. Three categories of Euclidean graphs are used: (a) graphs comprising of vertices generated using a uniform distribution in a two-dimensional plane (termed "CRD") with 30-100 vertices, (b) "SYM" graphs with 30-70 vertices generated in a higher dimensional Euclidean space (3–7 dimensions), and (c) "STR" graphs with vertices grouped as clusters and having 30 – 200 vertices. Four categories of non-Euclidean graphs are used: (a) "SHRD" – structured problem instances generated by assigning non-Euclidean distances to graph edges such that the number of optimal solutions is limited [13] on graph sizes of 15 – 150 vertices, (b) “Random” problem instances in which the weight of an edge is set using a uniform distribution (size from 15 to 1000 vertices), (c) “Random-hard” problems generated to contain a high degree of star patterns (50-200 vertices) and (d) “Misleading-hard” instances (50-500 vertex graphs) designed to mislead greedy algorithms.

The LACT algorithm is shown to obtain optimal solutions on some of the benchmark problems for which the optimal weight has been proven, and in general to obtain better solutions vis-a-vis several other well-known meta-heuristics on several benchmark problems. Other related works with brief description on degree-constrained minimum spanning tree problems are provided in Table 2.

2.1.1. Multi-Period Degree-Constrained Minimum Spanning Tree (MPDCMST)

Definition. MPDCMST is a process of connecting/installing vertices divide into periods that imply at a certain period only part of the set of vertices already connected, except the last period where all vertices must be connected. It is a problem of finding the total minimum installation cost whilst also maintaining the maximum number of edges incident to every vertex by Wamiliana et al. [14]. The restriction of the links on every vertex occurs to keep the reliability of the tree.

For example, suppose that City X does not have a freshwater network that connects entire buildings and houses in that city. The government of City X needs to build a pipe water network as a service to its people, but they have some constraints such as:

- Not all buildings can be connected in one period due to fund limitation (need more than one period, multi periods)
- Some buildings must be installed first (for example hospitals and other public facilities, and so on).
- The connection on some buildings cannot exceed a certain number of spots (degree restriction/constrained).
- All buildings must be installed when all period is over and installation finish but with the possible minimum cost (*minimum spanning tree***)**

The MPDCMST problem first introduced by Kawatra R. [15] who proposed a hybrid method between Lagrangian Relaxation and branch exchange, and used 10-year planning horizon and the period for activating each terminal is uniformly distributed. They implemented the method using graphs with orders varying from 40 to 100 vertices.

The modification of the MPDCMST problem is proposed by Wamiliana et al. [16]. They changed the planning horizon to one year and divided the installation process into three periods. They tested their method using a graph with vertex order from 10 up to 100 taken from by Wamiliana et al. [17] and all problems are feasible. Junaidi et. al. [18] improved the algorithm developed by Wamiliana et al. [19] and tested the algorithm on some problems taken from TSPLIB, and modified the method developed by setting HVTi = 3 for every i.

2.1.2. Bounded-Degree Spanning Tree (BDST)

Definition. The Euclidean degree-K MST problem is NP-hard for K ϵ {2, 3, 4}. It is well-known that one can compute a degree-2 spanning tree with weight at most 2 times the MST weight, by doubling the MST edges, computing an Euler tour, and then short-cutting repeated vertices. The constant 2 is tight, as Biniaz et al. [20] showed that for any fixed $\varepsilon > 0$ there exist point sets whose degree-2 MST weight is not smaller than $2 - \varepsilon$ times the MST weight. Therefore, T2 = 2.

In 1984, Papadimitriou and Vazirani [21] asked whether the Euclidean geometry can be exploited to obtain factors better than 2 for degree-3 and degree-4 spanning trees. Following this question, in 1994, Khuller et al. [22] achieved the following bounds for K = 3, 4: $1.103 < T3 \leq 1.5$, and $1.035 < T4 \leq 1.25$ respectively. The lower bounds are achieved by the center plus vertices of a square and a regular pentagon. The upper bounds are obtained by a recursive algorithm that roots the MST at a leaf v and then transforms it to a degree-K spanning tree with the inductive hypothesis that the root should have a degree at most K - 2 in the new tree. The algorithm transforms the subtrees rooted at the children of v recursively and then replaces the star (formed by v and its children) by a small-weight path in which v has a degree at most K - 2. A similar approach was studied before by Ravi et al. [23] for degree-3 spanning trees in metric spaces. In 2003, Chan [24] revisited the ratios and showed that the upper bounds 1.5 and 1.25 are almost tight if we insist on the root to have a degree at most K - 2. He managed to improve the corresponding upper bounds to $T3 < 1.402$ and $T4 < 1.143$ by a weaker inductive hypothesis that the root can have a degree at most K - 1 in the new tree, and by recursing not only on subtrees of the original MST but on trees formed by joining subtrees. A detailed analysis by Jothi et al. [25] shows that Chan's construction of degree-4 spanning trees gives upper bound $T4 < 1.1381$. Fekete et al. [26] conjectured that the lower bounds are tight, which is $T3 \approx 1.103$ and $T4 \approx 1.035$.

A natural open problem is given by Biniaz at al. [20] to improve the upper bounds on $\beta 2$, $\beta 3$, $\beta 4$ further by designing better algorithms. Specifically, it is natural to investigate whether the geometry of the Euclidean plane (besides the triangle inequality) can be exploited to develop an approximation algorithm for the bottleneck degree-2 spanning tree problem (i.e., the bottleneck Hamiltonian path problem) with factor less than 3. The existence of a 2-approximation algorithm given by Parker et al.

[27] for the bottleneck Hamiltonian cycle problem in general metric spaces; this algorithm is based on the Hamiltonicity of the square of every biconnected graph given by Fleischner et al. [28]. The study of worst-case ratios in higher dimensions is more vital as the maximum degree of an MST and a BST can be much larger. Zbarsky [29] showed that $1.447 \leq T3 \leq 1.559$ in the d-dimensional Euclidean space for any $d \geq 2$. Andersen et al. [30] studied the bottleneck version of the problem in dimension 3.

2.1.3. Minimum-Degree Constrained Minimum Spanning Tree (md-CMST)

Definition. The md-MST problem is closely related to the well-known degree-constrained minimum spanning tree problem given by Knowles et al. [31] where the degree of each vertex in the spanning tree should not exceed a given positive integer constant. In literature, there also exists other degree-dependent spanning trees such as the minimum branch vertices problem and the minimum degree sum of branch vertices given by Silvestri et al. [32]. Recently, another closely related version of the md-MST problem is introduced with fixed central and terminals, where the central and terminal nodes are fixed a priori.

Almeida, Martins, and de Souza [33] introduced the md-MST problem and proved that the problem is *NP*-hard for $\frac{n}{2} > d \geq 3$. Besides its computational complexity, the md-MST problem finds applications in the optimal design of networks (such as in VLSI design, road systems, energy, and computer communication networks), where the traffic between every pair of nodes is usually minimal. In such networks, centralized information devices can be installed to those special nodes whose degrees are greater than a given minimum value. Also proposed integer programming formulations based on single and multi-commodity flows for this problem and solved these formulations using branch-and-bound algorithms. Akgun et al. [34] proposed a new formulation via Miller–Tucker–Zemlin constraints and solved it through the branch-and-bound algorithm. Martinez and Cunha [35] presented other formulations for this problem and proposed a branch-and-cut algorithm based on the original directed formulation. Also, several metaheuristic techniques have been developed for this problem. Martins and de Souza [36] presented four versions of VNS approaches (VNS/ESO, Rk-VNS/ESO, Rk VNS/ESO1, Rk VNS/greedy). The first version (VNS/ESO) combines an enhanced version of a second-order repetitive technique (ESO) with the well-known VNS, where ESO is a repetitive technique arising from

the Second Order (SO) method and has been used within the improvement phase of the VNS framework, substituting the most common local search methodology. The second (Rk-VNS/ESO) and the third (Rk-VNS/ESO1) versions are randomized versions of the previous combination. The fourth version (Rk-VNS/greedy) just uses the greedy heuristic based on Kruskal's algorithm to explore each neighborhood in the VNS framework. Murthy and Singh [37] presented two swarm intelligence techniques, i.e. artificial bee colony algorithm and ant colony optimization algorithm, and they also tested their approaches on Euclidean and random instances. Salgueiro et al. [38] presented three versions of the generational genetic algorithm (*gen0*, *gen1*, and *gen2*) which differ in their chromosome representation and by the way of applying uniform crossover operator and mutation operator.

2.2. Bounded-Diameter Minimum Spanning Tree (BDMST) Problem

Definition. Given a connected, weighted, undirected graph G and an integer bound D, a bounded-diameter spanning tree (BDST) is a spanning tree on G whose diameter does not exceed D (the diameter of a tree is defined as the maximum number of edges along any path in the tree). The BDMST Problem aims to find a BDST on G of minimum weight.

The BDMST problem is known to be NP-hard for $4 \leq D < n - 1$. An exact algorithm for the Bounded-Diameter Minimal Spanning Tree (BDMST) Problem is given by Achuthan and Caccetta in [39] and an improved version that uses a branch and bound framework utilizing different branching rules and simple heuristics in. Gouveia and Magnanti [40], explore several variants of multi-commodity flow (MCF) formulations for the BDMST problem in which they count and limit the hops on paths from a virtual root node to any other node. While this model achieves extremely tight LP bounds (within 1% of the optimal solution for almost all benchmarks tested), it is only able to solve BDMST instances of up to 60 node graphs to optimality.

Since exact algorithms can solve only small problem instances, heuristics and meta-heuristic techniques have also been developed, which approximate low-cost BDSTs on much larger problem sizes within a reasonable time. Ayman Abdalla and Narsing Deo describe [41], several construction heuristics for the BDMST problem, including a Prim's algorithm-based heuristic called the *one-time tree construction* (OTTC) heuristic that runs in $O(n^4)$ time and produces low-cost BDSTs when the diameter bound D is small. Abdalla and Deo also give two iterative refinement algorithms designed for

somewhat "loose" diameter bounds. These algorithms start with an unconstrained MST and iteratively decrease the length of long paths until the diameter constraint is satisfied. Deo and Abdalla [42] give results for all these algorithms implemented on a massively parallel SIMD machine (single instruction, multiple data) with more than 8000 processors.

The *center-based tree construction* (CBTC) heuristic given by Julstrom in [43] performs better than OTTC both in terms of solution quality and running time (it requires O(n) timeless than OTTC) by constructing the BDST as a height-restricted tree rooted at a central node (or two nodes if the diameter is odd). A *randomized tree construction heuristic* (RTC) is also given in wherein each next node to be appended to the BDST is chosen at random and appended greedily. RTC works well on Euclidean problem instances and appears to be a good choice to quickly create a first reasonable solution as a basis for exact or meta-heuristic approaches. Gruber et al. present some hierarchical clustering-based heuristics for building low-cost BDSTs. Another recent heuristic is the *Center-based Least Sum-of-Costs* (CBLSoC) heuristic given by Patvardhan et al. [44] builds a low-cost BDST by repeatedly appending the non-tree vertex with the lowest mean cost to all the remaining non-tree vertices in the graph. This heuristic is run starting from every graph vertex and returns the lowest cost BDST obtained. It has a running time of $O(n^3)$ and performs competitively vis-a-vis the other heuristics.

Various evolutionary algorithms (EAs) have been developed for the BDMST problem to obtain better results. Raidl and Julstrom [45] present an EA that encodes candidate BDSTs as lists of their edges and uses recombination that produces offsprings having most edges in common with their parents. This edge-list coded EA also uses four fast mutation operators that produce new candidate solutions in almost linear time.

Julstrom and Raidl also propose a generational EA in [46] that encodes chromosomes as permutations of graph vertices, the permutation being the order in which the nodes are greedily appended to the BDST. This EA uses swap mutation and partially mapped (PMX) crossover, which tends to preserve the positions of nodes from the two parents to the offspring. This approach leads to better solutions but requires more time because decoding each permutation-coded chromosome requires $O(n^2)$ time.

Alok Singh et al. [47] improve the permutation encoded EA by including a local improvement given by Gruber et al. [48] in their decoding procedure which leads to better results in lesser time. A hybrid approach combining VNS with ACO and GA is also presented. A novel multi-parent recombination is proposed by Binh et al. in [49]. A GRASP (greedy randomized adaptive search procedure) and ILS (iterated local search) – based hybrid heuristic is given. A learning automata-based algorithm that rewards BDSTs if they have low cost and penalizes them otherwise is proposed by J. A. Torkestani [50].

The benchmark problems used in the literature comprise of several Euclidean and non-Euclidean graphs. One set of Euclidean problems used are taken from the Euclidean Steiner dataset in Beasley's OR-library [51], which comprise of completely connected graphs whose vertices are point coordinates in the unit square and whose edge weights are the Euclidean distances between each pair of vertices. Julstrom [43] augments these with one more set of completely connected Euclidean graphs and also gives a set of non-Euclidean complete graphs whose edge weights are chosen at random on the interval [0.01, 0.99]. Heuristics and meta-heuristics in the literature have been tested on benchmark graphs of up to 1000 vertices.

A hybrid tree construction heuristic is given by V. Prem Prakash et al. [52] that combines a greedy approach with a heuristic strategy for constructing effective tree "backbones". The performance of the proposed heuristic is shown to be consistently superior to the other extant heuristics on a standard benchmark suite of dense Euclidean graphs widely used in the literature.

Novel Hybrid Multi-objective Evolutionary Algorithm (HMOEA) for the bi-MDCST problem given by V. Prem Prakash et al. [53]uses a representation that captures the genetic information of several spanning trees of varying diameters in a single chromosome, results obtained by HMOEA are also reported for large-sized Euclidean instances which contain up to 500 vertices and 124,750 edges. Other related works with brief description on bounded-diameter minimum spanning tree problems are provided in Table 3.

2.2.1. bi-Objective Minimum Diameter-Cost Spanning Tree (bi-MDCST) Problem

Definition. Given a connected, weighted, undirected graph G with edge weights (or costs) w_{ij} associated with every edge $(i,j) \in E$, the bi-objective Minimum Diameter-Cost Spanning Tree (bi-MDCST)

Problem seeks a Pareto-optimal set of spanning trees on G that simultaneously minimize two conflicting objectives: the total weight of a spanning tree *T* on *G*, and the diameter of *T*. The bi-MDCST problem is NP-hard [54] and finds application in network design and transportation.

A multi-commodity flow formulation for the bi-MDCST problem is proposed in [55]. An exact algorithm for the problem is given in [56]. Results are reported on small complete graphs having at most 20 vertices, and a sparse graph instance having 40 vertices. The high computational time required by such approaches motivates the development of fast heuristics that return reasonably good solutions. Kumar and Saha [57] present some heuristics for the problem. The CBTC heuristic [58] is also recast for the bi-MDCST problem by Kumar and Saha [59], and the results obtained by the heuristics are improved further using an edge-list coded multi-objective evolutionary algorithm. Results are reported for complete random graphs having up to 100 vertices and complete Euclidean graphs having up to |V|=250 vertices. Improved heuristics for large Euclidean instances of the problem are reported in Prakash et al. [52]. An effective meta-heuristic for the problem that uses a novel chromosomal representation is presented in Prakash et al. [53].

2.3. Minimum Label Spanning Tree

Definition. Given a labeled graph G = (V, E, L), where V is the set of n nodes, E is the set of m edges, and L is the set of *l* labels, find a spanning tree T of G such that $|L_T|$ is minimized, where L_T is the set of labels used in T. It may be noted that edges' labels need not be unique; several edges may share a label.

The MVCA heuristic begins with an empty graph and then adds the label whose edges cover as many isolated vertices as possible. The heuristic solution is an arbitrary spanning tree of the resulting graph. This heuristic does not always yield a connected graph. Krumke et al. [60] propose a better version of MVCA, which begins with an empty graph and successively adds one label at random from amongst those labels that result in the least number of connected components. The procedure terminates when only one connected component is left, thus ensuring that a connected subgraph is always returned. Meta-heuristics based on local search have been applied to the MLST problem; for instance, Consoli et al. [61] use GRASP and VNS based approaches, and Cerulli et al. [62] apply Simulated Annealing, VNS, and Reactive Tabu Search to the problem. A genetic algorithm (GA) for the MLST problem that

encodes feasible solutions as lists of labels is given by Xiong et al. [63]. A modified GA by Xiong et al. [64] uses crossover that is performed by considering the combined list of labels of two individuals and then applying MVCA on this particular set of labels. The modified GA has longer running times but obtains better solutions.

Julstrom and Nummela [65] propose a GA that encodes candidates as permutations of labels. Decoding and evaluation are performed by iterating through the permutation, adding as many labels as required to build a feasible solution. After evaluation, a redundancy check is performed, reordering redundant labels immediately after the feasible ones. Crossover is performed by alternatively taking labels from the parents, and then eliminating duplicate labels by always keeping the first occurrence. The mutation is performed by swapping the one value of the feasible part with another one of the infeasible parts of the permutation.

Chwatal and Raidl [66] give an ant colony algorithm for the MLST problem and use local search to improve the solutions obtained using artificial ants. The algorithm performs well on standard benchmark graphs of up to 500 vertices and 625 labels. In particular, when data instances do not contain single isolated optimums, but rather numerous optimal solutions, the method reliably finds the global optimum within relatively short computational time. A recent work of Consoli et al. [67] integrates VNS with other complements from machine learning, statistics, and experimental algorithmics. The resultant hybrid meta-heuristic is shown to perform better than other approaches currently in the literature and obtain optimal or near-optimal solutions in short computational running time on the benchmark instances. Other related works with brief description on minimum label spanning tree problems are provided in Table 4.

2.4. Leaf-Constrained Minimum Spanning Tree Problem

Definition. Let L be an integer such that $2 \leq L < n - 1$. A leaf constrained spanning tree (LCST) is a spanning tree on G that has at least L leaves, and a leaf-constrained minimum spanning tree (LCMST) is an LCST of minimum total weight. The Leaf-Constrained Minimum Spanning Tree Problem searches for an LCMST with at least L leaves on a given graph G.

Deo and Micikevicius [68] show this problem to be NP-hard and give a greedy heuristic that runs in $O(n^4)$ time. This heuristic starts with an unconstrained MST and exchanges tree edges with non-

tree edges in such a way as to increase the number of leaves while keeping the increase in tree cost/weight as less as possible. The heuristic sometimes fails to produce a valid tree.

Julstrom [69] gives a better performing greedy heuristic called ML (an acronym for "more leaves") and two-generational genetic algorithms. The heuristic (ML) begins by computing an MST and then iteratively transforming it into an LCST. It then iteratively selects an interior vertex (a vertex which is not a leaf) for converting it into a leaf according to the following criteria (it tries each interior vertex as a leaf one by one): During each trial, it computes an MST on the graph induced by remaining interior vertices then connects all leaves including the new one to the nearest interior vertex. The interior vertex for which the resulting spanning tree is of the smallest weight is selected to become a leaf. This process is repeated until the number of leaves in the tree reaches L. ML always returns an LCST if the underlying graph is complete. One of the generational GAs proposed by Julstrom uses Blob Code [70] and the other uses subset coding [70]. The subset coded GA (SCGA) represents an LCST by its interior vertices only; a two-step procedure converts this coding into its equivalent LCST. The first step constructs an MST on the interior vertices, whereas the latter step joins each leaf to its nearest interior vertex. Both GAs is designed in such a way that they produce only feasible solutions. On a wide range of Euclidean and random benchmark problems, the ML heuristic is shown to perform better than the Blob-Coded GA, while the SCGA consistently obtains the lowest weight LCSTs.

Alok Singh et al. [71] give an ant-colony optimization (ACO) algorithm (ACO-LCMST) and a Tabu-Search based algorithm (TS-LCMST) that use the subset coding from Julstrom's GAs [69] to represent the MLST. Alok Singh [72] also gives an artificial bee colony (ABC) algorithm for the problem (ABC-LCMST), also using the same subset coding-based chromosomal representation, that obtains better quality solutions in a shorter time than the ACO and Tabu search algorithms.

A recent Modified Quantum-behaved PSO algorithm (MQPSO) that amalgamates quantum mechanics concepts and a heuristic modification to avoid local optima in the PSO algorithm is given by Farzi et al. [73]. This modification tries to avoid local optimum by defining a "leaping" behavior in the QPSO that helps move quickly from local optimum to global optimum and improve the global convergence of QPSO. The performance of the MQPSO is compared with extant algorithms like ML, SCGA, ACO-LCMST, TS-LCMST, and ABC-LCMST on sixty-five standard benchmarks. Forty-five

of these instances are Euclidean graphs: fifteen instances each of 50, 100, and 250 vertices. The remaining twenty instances are non-Euclidean: ten instances each of 100 and 300 vertex graphs. The Euclidean problem instances are taken from Beasley's OR-Library [51] and the non-Euclidean problems from Julstro [69] m. The MQPSO approach obtains better quality solutions in lesser time as compared to the other recent meta-heuristics on the benchmark problems.

Two novel heuristics are given by V. Prem Prakash et al. [74] that obtain good solutions for the e-LCMST problem in O(|V|3) time and scale well with larger problem sizes. In this work, A relative comparison of the average execution times reported for the ACO-LCMST, TS-LCMST, ABC-LCMST, KCB, and LSoCV is visualized for 250 vertex complete graphs. Other related works with brief description on Leaf-constrained minimum spanning tree problems are provided in Table 5.

2.5. Optimum Communication Spanning Tree Problem

Definition. Given a graph G = (V, E, d, r) with distance function d: E→$\mathbb{R}^+$ and requirement function r: V×V → $\mathbb{R}^+$, find a spanning tree T ⊆ G that minimizes the cost c(T) = $\sum_{i,j\in V} r(i,j).\, c(p_{i,j}^T)$, where c($p_{i,j}^T$) is the length of path $p_{i,j}^T$ from i to j in T.

The Optimum Communication Spanning Tree (OCST) Problem first appears in T.C. Hu [75]. The problem is known to be NP-hard [3]. Presents two special cases of this problem: The *Optimum Requirement Spanning Tree Problem* (d=1), which can be solved in O(n^4) time using a construction algorithm that uses a Ford-Fulkerson-based labeling procedure, and the *Optimum Distance Spanning Tree Problem* (r=1). Some other special cases of the OCST Problem and some approximation algorithms are given by Wu et al. [76]: these include the *Product-requirement Communication Spanning Tree Problem* (PROCT, r(i, j) = r(i) · r(j)) and the *Sum-requirement Communication Spanning Tree Problem* (SROCT, r(i, j) = r(i) + r(j)). Approximation algorithms are also given for PROCT (O(n^5) time, 1.577-approximation), SROCT (O(n^3) time, 2-approximation), and MRCT (O(n^3), 1.577-approximation). An exact branch-and-bound algorithm is given by Ahuja and Murty in [77], in which the lower bound is computed in O(n^4) time and hence unsuitable for large problem instances.

A genetic algorithm is given by Palmer et al. [78] in which chromosomes contain a vector of bias values for both nodes and edges modifying the edge cost function. The tree represented by the chromosomes is constructed by building an MST using the modified cost function.

A better performing GA is given by S.M. Soak [79], in which spanning trees are represented as a sequence of node IDs, where two consecutive nodes form an edge that is included in the tree under construction. Once a cycle is introduced, the longest edge is removed from the cycle and the construction of the tree continued. The crossover starts from a random node and adds edges to the node-sequence that occur in one of the parents.

Other genetic algorithms have been proposed for the problem that uses different tree representations and search operators, such as characteristic vector (CV) encoding [80], weighted encoding [81], edge-sets [82], NetKeys encoding, and Prüfer numbers [83]. Fischer and Merz give a memetic algorithm in [84] that performs well on larger problem sized than hitherto before. They give results on Euclidean graphs of up to 100 vertices and non-Euclidean graphs on up to 500 vertices. Hoang et al. [85] give a PSO algorithm that uses node biased encoding, and a Hybrid Genetic Algorithm (HGA) that combines the basic genetic algorithm with PSO is given by Kien et al. [86]. Experimental results on benchmark instances show that the HGA outperforms the other GAs mentioned, both in terms of convergence and solution quality. Other related works with brief description on optimum communication minimum spanning tree problems are provided in Table 6.

3. Parallelization Approach

For several years now, the scope of performance improvements in single-core processors has been affected by semiconductor scaling limits and associated thermal issues and power requirements (the "Power Wall"), combined with the difficulty in exploiting greater levels of instruction-level parallelism (the "ILP Wall"). This has led, over the past decade, a major approach in microprocessor design. The multi-core approach, wherein the processor speed remains the same but the number of processor cores is increased.

Several parallel algorithms for computing minimum spanning trees appear in the literature, R. Helgaon et al. [87], F. Dehne et al. [88], and others. A recent work by Gonina and Kale [89] describes a parallel algorithm based on Prim's algorithm that adds multiple vertices in each iteration by generating

an array of the K graph vertices closest to the tree and then performing a global reduction to obtain an array of K vertices globally closest to the current partial MST. The algorithm, implemented using MPI, achieves good improvements in performance as demonstrated on large-sized dense graphs of up to 200,000 vertices.

Rohit Setia et al. [90] give a parallel Prim-based algorithm for building minimum spanning trees that target shared-memory SMPs. The algorithm non-deterministically chooses a node as root, and multiple threads grow a tree in parallel by coloring the nodes with a unique color, called its ID. Threads communicate asynchronously, and if a thread wishes to add a node that belongs to another sub-tree, the two corresponding sub-trees are merged. A Merge Tree operation forces the tree grown by the thread with a higher thread ID to merge with the tree grown by the thread with a lower thread ID so that eventually thread 0 has the MST. This algorithm is shown to achieve reasonable speedup in comparison to Serial Prim's algorithm for a wide range of dense and sparse graphs.

Parallel computing using accelerators has gained widespread research attention in the past few years. In particular, the use of Graphics Processing Units (GPUs) for general-purpose computing has resulted in dramatic performance improvements in terms of the time taken, cost, power, and other metrics. A recent work by Vineet et al. [91] uses scalable data mapping primitives to implement an efficient parallel implementation of a minimum spanning tree algorithm on NVidia GPUs under CUDA as a recursive formulation of Boruvka’s algorithm for undirected graphs. The authors show a speedup of 30 to 50 times over the CPU implementation on most graphs and demonstrate the scalability of their approach by building a minimum spanning tree on a 5 million node and 30 million edge graphs in under 1 second on one-quarter of the Tesla S1070 GPU. Wang et al. [92] propose a CUDA-based parallel version of Prim's minimum spanning tree algorithm that efficiently builds MSTs using GPU-based Min-Reduction data-parallel primitives. Scott Rostrup et al. [93] present a memory-efficient data-parallel implementation of Kruskal's algorithm for finding the minimum spanning tree of very large graphs and demonstrate the performance of their algorithm on random and real-world graphs with up to 25 million vertices and 240 million edges on an NVidia Tesla T10 GPU with 4GB of memory.

McCreesh et al. [94] describes constraint programming and several parallel methods concentrated on how to solve the subgraph isomorphism problem. Archibald et al. [95] designed an

algorithm for subgraph isomorphism that instead of using the degree of a vertex to sort the matching pairs, adopts it to direct the proportion of search effort spent in different subproblems. This increases parallelization which helps to get good speedups on both satisfiable and unsatisfiable instances. Quer et al. [96] compare their structural decomposition with domain-based decompositions, which split variable domains. Finally proved that the new method leads to better speedups in some classes of instances. McCreesh et al. [97] present parallel algorithms for the maximum clique and the subgraph problem. Hoffmann et al. [98] compare the sequential and parallel versions of three state-of-the-art branch and bound algorithms. Quer et al. [96] discusses the possibility to extend an existing, very effective branch-and-bound procedure on parallel multi-core and many-core architectures. Some other works concentrate on similar but different problems. For example, Kimmig et al. [99] study the subgraph enumeration problem, in which the target is to find all subgraphs of a target graph that are isomorphic to a given pattern graph. The authors present a shared-memory parallelization of state-of-the-art subgraph enumeration algorithms based on work-stealing. Their implementation demonstrates a significant speedup on real-world biochemical data, despite a highly irregular data access pattern. The maximum clique problem is the subject of McCreesh et al. [97].

Prem Prakash et al. [100] proposes two-phase memetic algorithm for the BDMST problem that combines a specialized recombination operator with good heuristics in order to more effectively direct the exploration of the search space into regions containing better solutions. A parallel meta-heuristic is also proposed and shown to obtain very good speedups.

Recently work done by Stefano et al. [96] extends a state-of-the-art branch-and-bound algorithm used to find the maximum common subgraphs between two graphs in three main directions. First, they parallelize it into a multi-core CPU implementation and a CUDA-based GPU many-core one. Concerning the CPU procedure, they experiment with several parallel approaches, going from using different data structures to adopting several divide-and-conquer paradigms, to balance the threads workload and to keep under control the memory usage. As far as the CUDA extension is concerned, like many other graph algorithms, the analyzed one is based on complex data structures and recursion. Both features do not usually suit well with GPU computing, and they propose a few ways to overcome these limitations. Second, they develop new heuristics to reorder the adjacency matrix, trying to isolate

matrix blocks and make them upper triangular, to smooth the heavy-tail phenomenon dealing with dead-ends, and to restart the process with automatic restarts. These strategies allow a significant performance improvement for specific graph pairs. Finally, they propose a portfolio approach, integrating several local search algorithms as component tools. In this portfolio, a python interface orchestrates many CPU versions with one GPU procedure, exploiting modern hardware resources at their best. They demonstrate that the portfolio approach is beneficial in the domain, and CUDA versions may worth further analysis.

These results show that there is significant scope for the development of effective parallel algorithms and parallel meta-heuristics for other constrained COPs in graphs as well, and on harnessing the power of GPGPUs for solving such problems. Some well-known parallel programming models such as MPI and CUDA are briefly described as follows.

3.1. MPI/Open MPI

The message-passing model demonstrates the following characteristics: A set of tasks that use their local memory during computation. Multiple tasks can reside on the same physical machine as well across an arbitrary number of machines. Tasks exchange data through communications by sending and receiving messages. Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation. From a programming perspective, message passing implementations commonly comprise a library of subroutines that are embedded in source code. The programmer is responsible for determining all parallelism. Historically, a variety of message passing libraries have been available since the 1980s. These implementations differed substantially from each other making it difficult for programmers to develop portable applications. In 1992, the MPI Forum was formed with the primary goal of establishing a standard interface for message passing implementations [101].

3.2. CUDA (Compute Unified Device Architecture)

A current multi-core processor like the Intel Core i9 microprocessor supports Hyper-threading, Turbo Boost with 8 cores, 16 threads, and is designed to maximize the execution speed of sequential programs. The other major trend is the many-core approach, which focuses more on the execution throughput of parallel applications. Many-core systems typically consist of a large number of much smaller cores. The

NVIDIA GeForce RTX 3090 graphics processing unit (GPU) is an instance of a many-core processor; it has 82 SMs, 10496 CUDA cores, 82 RT cores, and 328 Tensor cores. In general-purpose computation using GPUs (GPGPU), the GPU is viewed as a massively multi-threaded architecture containing several pipelined processor cores. The GPU architecture fits the data-parallel computing model well, with a single processing kernel applied to a large data grid.

While the performance improvement of general-purpose microprocessors has slowed significantly, the GPUs have continued to improve relentlessly, leading to a large performance gap between parallel and sequential execution. This performance gap has already motivated many application developers to move the computationally intensive parts of their software to GPUs for execution and possible parallelization. GPUs also offer extensive resources for general-purpose computations: massive parallelism, high memory bandwidth, and general-purpose instruction sets, including support for both single- and double- precision IEEE floating-point arithmetic. Before the advent of general-purpose languages for GPGPU, GPU implementations could only be achieved using existing 3D-rendering APIs like OpenGL or DirectX. The syntax and the need to pose problems in the context of polygon rasterization made this a rather difficult approach. Recognizing the potential for GPU computing, GPU vendors like NVIDIA, AMD and others started adding driver and hardware support to use the highly parallel hardware of the GPU without the need for computation to proceed through the entire graphics pipeline or using Direct3D APIs at all. NVIDIA's solution is the Compute Unified Device Architecture (CUDA) language, an extension to C/C++ based on some fundamental parallel programming abstractions. NVIDIA also provides the CUDA C/C++ compiler, libraries, and other development tools and is currently working to support other programming languages in general purpose and scientific computing like Fortran and MATLAB.

4. Meta-heuristics

As mentioned in section 2, a vast majority of the COPs under consideration are NP-hard problems, and exact algorithms for solving them by enumerating all possible solutions are feasible only for very small problem sizes, as the running time becomes exponential in the input size. It is possible to improve these algorithms by performing the enumerations in a more intelligent manner using techniques like Linear Programming (LP)-based branch-and-bound, branch-and-cut, and other related techniques, but the

running time and memory requirements for exact algorithms remain prohibitive for anything more than small to moderate sizes of problem instances in most cases. Consequently, heuristics and meta-heuristics that can obtain good, even near-optimal solutions within reasonable computation times gain significance. Several meta-heuristic strategies have been developed and applied in the literature; these include (but are not limited to) Memetic Algorithm, Grey Wolf optimization, Metaphor-Inspired meta-heuristics, Simulated Annealing (SA), Tabu Search (TS), Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighbourhood Search (VNS), Evolutionary Algorithms (EAs), Quantum-inspired Evolutionary Algorithms (QGAs), Estimation of Distribution Algorithms (EDAs), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) Optimization. As we know the process to find feasible solutions is time taking process, sometimes it takes days to return the good results because of the big size population and lots of generations. So, several different approaches are developed to overcome such kind of problems, like parallel approaches that can handle the heavily workload-unbalance and the massive data dependency. These heuristics can achieve significant speed-ups in specific instances. The proposed approach drastically limits memory bandwidth constraints and avoids another typical portfolio fragility as CPU and GPU versions often show a complementary efficiency and run on a separate platform.

5. Proposed Work

The literature review given in the previous sections throws up many open issues. It is proposed to address some of these problems in the present work.

1. Some of the COPs that could be studied in this work are: Leaf-Constrained Minimum Spanning Tree Problem, Regenerator Location Problem, Multi-Period Degree Constrained Minimum Spanning Tree Problem and Diameter Constrained Minimum Spanning Tree Problem. Polynomial-time heuristics do exist for some of the problems discussed in the literature review but are typically of higher order viz., $O(n^3)$ or above. This limits the sizes of graphs on which these heuristics can be employed. Therefore, there is scope for further exploration of more efficient heuristics that obtain better solutions and/or scale well for solving much larger problem sizes than attempted hitherto.

2. Carefully hybridizing meta-heuristics with good heuristics is generally found to be effective in speeding up the convergence to the optimal solution thereby reducing the overall computation time. Attempts would be made to design novel memetic algorithms for the problems that would incorporate the good features of existing and/or proposed heuristics. Heuristics would be developed with problem solving as the goal. Using intuitive problem understanding, hybrid techniques would be developed and tested empirically.

3. The detailed literature survey shows that not much work has been done on the development of parallel heuristics and metaheuristics for these problems. Parallel programming models such as MPI, OpenMP and CUDA have been used in several domains for obtaining computational speedup and providing better quality solutions. Thus, there is substantial potential in harnessing the power of current multi- and many-core architectures for efficiently finding near optimal solutions to much larger instances. These would be explored for the problems at hand as applicable and effective. Parallel computing infrastructure is available: MPI, OMP, and CUDA. Implementation would be developed using one or more of these technologies as deemed suitable/effective.

6. References

[1] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation,* pp. 67-82, 1 1997.

[2] Y. C. Ho and D. L. Pepyne, "Simple explanation of the no free lunch theorem and its implications," *Journal of Optimization Theory and Applications,* vol. 115, no. 3, pp. 549-570, 2002.

[3] M. R. Garey and D. S. Johnson, Computers and Intractibility: A Guide to the Theory of NPCompleteness, New York: W.H. Freeman, 1979.

[4] G. Zhou and M. Gen, "A note on genetic algorithms for degree constrained spanning tree problems," *Networks,* vol. 30, no. 2, pp. 91-95, 1998.

[5] G. R. Raidl and B. A. Julstrom, “A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem,” in *Proceedings of the 2000 ACM symposium on Applied computing-Volume 1*, 2000.

[6] M. d. Souza and P. Martins, “Skewed VNS enclosing second order algorithm for the degree constrained minimum spanning tree problem,” *European Journal of Operations Research,* vol. 191, pp. 677-690, 2008.

[7] P. Martins and M. d. Souza, “VNS and second order heuristics for the min-degree constrained minimum spanning tree problem,” *Comput Oper Res,* vol. 36, p. 2969–2982, 2009.

[8] Y.-T. Bau, H. C-K and H.-T. Ewe, “Ant colony optimization approaches to the degree-constrained minimum spanning tree problem,” *J Inf Sci Eng,* vol. 24, no. 4, p. 1081–1094, 2008.

[9] T. Bui, X. Deng and C. Zrncic, “An improved ant-based algorithm for the degree-constrained minimum spanning tree problem,” *IEEE Transactions on Evolutionary Computation,* vol. 16, no. 2, pp. 266-278, 2012.

[10] A. Ernst, “A hybrid Lagrangian particle swarm optimization algorithm for the degree constrained minimum spanning tree problem,” in *IEEE Congress on Evolutionary Computation*, 2010.

[11] D. Wedelin, “An algorithm for large scale 0-1 integer programming with application to airline crew scheduling,” *Annals of Operations Research,* vol. 57, pp. 283-301, 1995.

[12] J. A. Torkestani, “Degree constrained minimum spanning tree problem: a learning automata approach,” *Journal of Supercomputing,* vol. 64, pp. 226-249, 2013.

[13] M. Krishnamoorthy and A. Ernst, “Comparison of algorithms for the degree constrained minimum spanning tree,” *Journal of Heuristics,* vol. 7, pp. 587-611, 2001.

[14] W. Wamiliana, M. Usman, D. Sakethi, R. Yuniarti and A. Cucus, “The hybrid of depth first search technique and Kruskal_s algorithm for solving the multiperiod degree constrained minimum spanning tree problem,” 2015.

[15] R. Kawatra, "A multiperiod degree constrained minimal spanning tree problem," *European Journal of Operational Research,* vol. 143, p. 53–63, 2002.

[16] Wamiliana, C. U. of Technology. Dept. of Mathematics and Statistics, *Combinatorial methods for dgree constrained minimum spanning tree problem,* 2002.

[17] W. Wamiliana, "Solving the degree constrained minimum spanning tree problem using tabu and modified penalty search methods," *Jurnal Teknik Industri,* vol. 6, p. 1–9, 2004.

[18] A. Junaidi, W. Wamiliana, D. Sakethi and E. T. Baskoro, "COMPUTATIONAL ASPECTS OF GREEDY ALGORITHM FOR SOLVING THE MULTI PERIOD DEGREE CONSTRAINED MINIMUM SPANNING TREE PROBLEM," *Jurnal Sains MIPA Universitas Lampung,* vol. 2, 2012.

[19] W. Wamiliana, D. Sakethi, A. Junaidi and E. T. Baskoro, "THE DESIGN OF GREEDY ALGORITHM FOR SOLVING THE MULTI PERIOD DEGREE CONSTRAINED MINIMUM SPANNING TREE PROBLEM," *Jurnal Sains MIPA Universitas Lampung,* vol. 3, 2012.

[20] A. Biniaz, "Euclidean bottleneck bounded-degree spanning tree ratios," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms,* 2020.

[21] C. H. Papadimitriou and U. V. Vazirani, "On two geometric problems related to the travelling salesman problem," *Journal of Algorithms,* vol. 5, p. 231–246, 1984.

[22] S. Khuller, B. Raghavachari and N. Young, "Low-degree spanning trees of small weight," *SIAM Journal on Computing,* vol. 25, p. 355–368, 1996.

[23] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz and H. B. Hunt III, "Many birds with one stone: Multi-objective approximation algorithms," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing,* 1993.

[24] T. M. Chan, "Euclidean bounded-degree spanning tree ratios," *Discrete & Computational Geometry,* vol. 32, p. 177–194, 2004.

[25] R. Jothi and B. Raghavachari, “Degree-bounded minimum spanning trees,” *Discrete Applied Mathematics,* vol. 157, p. 960–970, 2009.

[26] S. P. Fekete, S. Khuller, M. Klemmstein, B. Raghavachari and N. Young, “A network-flow technique for finding low-weight bounded-degree spanning trees,” *Journal of Algorithms,* vol. 24, p. 310–324, 1997.

[27] R. G. Parker and R. L. Rardin, “Guaranteed performance heuristics for the bottleneck travelling salesman problem,” *Operations Research Letters,* vol. 2, p. 269–272, 1984.

[28] H. Fleischner, “The square of every two-connected graph is Hamiltonian,” *Journal of Combinatorial Theory, Series B,* vol. 16, p. 29–34, 1974.

[29] S. Zbarsky, “On improved bounds for bounded degree spanning trees for points in arbitrary dimension,” *Discrete & Computational Geometry,* vol. 51, p. 427–437, 2014.

[30] P. J. Andersen and C. J. Ras, “Degree bounded bottleneck spanning trees in three dimensions,” *Journal of Combinatorial Optimization,* vol. 39, p. 457–491, 2020.

[31] J. Knowles and D. Corne, “A new evolutionary approach to the degree-constrained minimum spanning tree problem,” *IEEE Transactions on Evolutionary computation,* vol. 4, p. 125–134, 2000.

[32] S. Silvestri, G. Laporte and R. Cerulli, “A branch-and-cut algorithm for the minimum branch vertices spanning tree problem,” *Computers & Operations Research,* vol. 81, p. 322–332, 2017.

[33] A. M. de Almeida, P. Martins and M. C. Souza, “md-MST is NP-hard for d= 3,” *Electronic Notes in Discrete Mathematics,* vol. 36, p. 9–15, 2010.

[34] İ. Akgün and B. Ç. Tansel, “Min-degree constrained minimum spanning tree problem: New formulation via Miller–Tucker–Zemlin constraints,” *Computers & Operations Research,* vol. 37, p. 72–82, 2010.

[35] L. C. Martinez and A. S. Da Cunha, “The min-degree constrained minimum spanning tree problem: Formulations and Branch-and-cut algorithm,” *Discrete Applied Mathematics,* vol. 164, p. 210–224, 2014.

[36] M. Pedro and d. S. M. C., “VNS and second order heuristics for the min-degree constrained minimum spanning tree problem,” *Computers & operations research,* vol. 36, p. 2969–2982, 2009.

[37] V. V. R. Murthy and A. Singh, “An ant colony optimization algorithm for the min-degree constrained minimum spanning tree problem,” in *International Conference on Swarm, Evolutionary, and Memetic Computing*, 2013.

[38] R. Salgueiro, A. de Almeida and O. Oliveira, “New genetic algorithm approach for the min-degree constrained minimum spanning tree,” *European Journal of Operational Research,* vol. 258, p. 877–886, 2017.

[39] N. Achuthan and L. Caccetta, “Minimum weight spanning trees with bounded diameter,” *Australasian Journal of Combinatorics,* vol. 5, p. 261–276, 1992.

[40] L. Gouveia and T. Magnanti, “Network flow models for designing diameter constrained minimum spanning and Steiner trees,” *Networks,* vol. 41, no. 3, p. 159–173, 2003.

[41] A. Abdalla, N. Deo and P. Gupta, “Random-tree diameter and the diameter constrained minimum spanning tree,” *Congressus Numerantium,* vol. 144, p. 161–182, 2000.

[42] N. Deo and A. Abdalla, “Computing a diameter-constrained minimum spanning tree in parallel,” in *Algorithms and Complexity*, 2000.

[43] B. Julstrom, “Greedy heuristics for the bounded diameter minimum spanning tree problem,” *Journal of Experimental Algorithmics (JEA),* vol. 14, no. 1, pp. 1-14, February 2009.

[44] C. Patvardhan and V. P. Prakash, “Novel deterministic heuristics for the bounded diameter minimum spanning tree problem,” in *Pattern Recognition and Machine Intelligence*, New Delhi, 2009.

[45] G. Raidl and B. Julstrom, “Greedy heuristics and an evolutionary algorithm for the bounded diameter minimum spanning tree problem,” in *ACM Symposium on Applied Computing*, 2003.

[46] B. Julstrom and G. Raidl, “A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem,” in *Genetic and Evolutionary Computation*

Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations, 2003.

[47] A. Singh and A. Gupta, "Improved heuristics for the bounded-diameter minimum spanning tree problem," *Soft Computing – A Fusion of Foundations, Methodologies and Applications,* vol. 11, no. 10, p. 911–921, 2007.

[48] M. Gruber, J. Hemert and G. Raidl, "Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA and ACO," in *Genetic and evolutionary computation conference*, Seattle, 2006.

[49] H. Binh and N. Nghia, "New multiparent recombination in genetic algorithm for solving bounded diameter minimum spanning tree problem," in *First Asian Conference on Intelligent Information and Database Systems*, 2009.

[50] T. J. Akbari, "An adaptive heuristic for the bounded diameter minimum spanning tree problem," *Soft Computing,* vol. 16, no. 11, pp. 1977-1988, 2012.

[51] J. Beasley, "OR-Library: distributing test problems by electronic mail," *Journal of the Operational Research Society,* vol. 41, no. 11, pp. 1069-1072, 1990.

[52] V. P. Prakash, C. Patvardhan and A. Srivastav, "Effective heuristics for the bi-objective euclidean bounded diameter minimum spanning tree problem," in *International Conference on Next Generation Computing Technologies*, 2017.

[53] V. P. Prakash, C. Patvardhan and A. Srivastav, "A novel Hybrid Multi-objective Evolutionary Algorithm for the bi-Objective Minimum Diameter-Cost Spanning Tree (bi-MDCST) problem," *Engineering Applications of Artificial Intelligence,* vol. 87, p. 103237, 2020.

[54] J.-M. Ho, D. T. Lee, C.-H. Chang and C. K. Wong, "Minimum diameter spanning trees and related problems," *SIAM Journal on Computing,* vol. 20, p. 987–997, 1991.

[55] A. C. Santos, D. R. Lima and D. J. Aloise, "Modeling and solving the bi-objective minimum diameter-cost spanning tree problem," *Journal of Global Optimization,* vol. 60, p. 195–216, 2014.

[56] E. G. De Sousa, A. C. Santos and D. J. Aloise, “An exact method for solving the bi-objective minimum diameter-cost spanning tree problem,” *RAIRO-Operations Research,* vol. 49, p. 143–160, 2015.

[57] S. Saha, M. Aslam and R. Kumar, “Assessing the performance of bi-objective MST for Euclidean and non-Euclidean instances,” in *International Conference on Contemporary Computing*, 2010.

[58] B. A. Julstrom, “Greedy heuristics for the bounded diameter minimum spanning tree problem,” *Journal of Experimental Algorithmics (JEA),* vol. 14, p. 1–1, 2010.

[59] S. Saha and R. Kumar, “Bounded-diameter MST instances with hybridization of multi-objective EA,” *International Journal of Computer Applications,* vol. 18, p. 17–25, 2011.

[60] S. Krumke and H.-C. Wirth, “On the minimum label spanning tree problem,” *Information Processing Letters,* vol. 66, no. 2, pp. 81-85, 1998.

[61] S. Consoli, A. M. J., N. Mladenovic and K. Darby-Dowman, “Heuristics based on greedy randomized adaptive search and variable neighbourhood search forthe minimum labelling spanning tree problem,” *European Journal of Operational Research,* vol. 196, p. 440–449, 2009.

[62] R. Cerulli, A. Fink, M. Gentili and S. Voß, “Metaheuristics comparison for the minimum labelling spanning tree problem,” *Springer Operations Research/Computer Science Interfaces Series,* vol. 29, pp. 93-106, 2005.

[63] Y. Xiong, B. Golden and E. Wasil, “A one-parameter genetic algorithm for the minimum labeling spanning tree problem,” *IEEE Transactions on Evolutionary Computation,* vol. 9, no. 1, pp. 55-60, 2005.

[64] Y. Xiong, B. Golden and E. Wasil, “Improved heuristics for the minimum label spanning tree problem,” *IEEE Transactions on Evolutionary Computation,* vol. 10, December 2006.

[65] B. Julstrom and J. Nummela, “An effective genetic algorithm for the minimum label spanning tree problem,” in *Eighth Annual Genetic and Evolutionary Computation Conference*, New York, 2006.

[66] A. Chvatal and G. Raidl, “Solving the Minimum Label Spanning Tree Problem by Ant Colony Optimization,” in *International Conference on Genetic and Evolutionary Methods (GEM 2010)*, Las Vegas, 2010.

[67] S. Consoli, J. M. Pérez and N. Mladenović, “Intelligent variable neighbourhood search for the minimum labelling spanning tree problem,” *Electronic notes on Discrete Mathematics*, vol. 41, pp. 399-406, June 2013.

[68] N. Deo and P. Micikevicius, “A heuristic for the leaf-constrained minimum spanning tree problem,” *Congressus Numerantium,* vol. 141, p. 61–72, 1999.

[69] B. Julstrom, “Codings and operators in two genetic algorithms for the Leaf-Constrained Minimum Spanning Tree Problem,” *Int. J. Appl. Math. Comput. Sci.,* vol. 14, no. 3, pp. 385-396, 2004.

[70] S. Picciotto, “How to encode a tree,” San Diego, 1999.

[71] A. Singh and A. Baghel, “New Metaheuristic Approaches for the Leaf-Constrained Minimum Spanning Tree Problem,” *Asia Pacific Journal of Operational Research,* vol. 25, no. 4, pp. 575-589, 2008.

[72] A. Singh, “An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem,” *Applied Soft Computing,* vol. 9, no. 2, pp. 625-631, 2009.

[73] S. Farzi and A. Dastjerdi, “Leaf constrained minimal spanning trees solved by modified quantum-behaved particle swarm optimization,” *Artifial Intelligence Review,* vol. 34, pp. 1-17, 2010.

[74] P. V. Prem and P. C., “Novel Heuristics for the Euclidean Leaf-Constrained Minimum Spanning Tree Problem,” *SN Computer Science,* vol. 1, p. 1–19, 2020.

[75] T. Hu, “Optimum Communication Spanning Trees,” *SIAM Journal of Computing,* vol. 3, no. 3, pp. 188-195, 1974.

[76] B. Wu, K. Chao and C. Tang, “Approximation algorithms for some optimum communication spanning tree problems,” *Discrete Applied Mathematics,* vol. 102, no. 3, pp. 245-266, 2000.

[77] R. Ahuja and V. Murty, “Exact and Heuristic Algorithms for the Optimum Communication Spanning Tree Problem,” *Transp. Sci.,* vol. 21, no. 3, pp. 163-170, 1987.

[78] C. Palmer and A. Kershenbaum, “Representing trees in genetic algorithms,” in *Proceedings of the First IEEE Conference on Evolutionary Computation,* 1994.

[79] S.-M. Soak, “A New Evolutionary Approach for the Optimal Communication Spanning Tree Problem,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,* Vols. E-89A, no. 10, pp. 2882-2893, October 2006.

[80] T. M. B. L., B. A. Murtagh and M. G., “Applications of a genetic-based algorithm for optimal design of tree-structured communication networks,” in *Proc. Regional Teletraffic Engrg. Conf. Internat. Teletraffic Congress,* Pretoria, 1995.

[81] G. Raidl and B. Julstrom, “A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem,” in *ACM Symposium on Applied Computing*, 2000.

[82] F. Rothlauf and C. Tzschoppe, “Making the edge-set encoding fly by controlling the bias of its crossover operator,” in *Evolutionary Computation in Combinatorial Optimization*, 2005.

[83] F. Rothlauf, D. Goldberg and A. Heinzl, “Network random keys – A tree network representation scheme for genetic and evolutionary algorithms,” *Evolutionary Computation,* vol. 10, no. 1, pp. 75-97, 2002.

[84] T. Fischer and P. Merz, “A Memetic Algorithm for the Optimum Communication Spanning Tree Problem,” in *4th International Workshop on Hybrid Metaheuristics*, Dortmund, 2007.

[85] A. Hoang, V. Le and N. Nguyen, “A Novel Particle Swarm Optimization-Based Algorithm for the Optimal Communication Spanning Tree Problem,” in *Second International Conference on Communication Software and Networks*, 2010.

[86] P. Kien, N. Hiep and H. Binh, “New Hybrid Genetic Algorithm for Solving Optimal Communication Spanning Tree Problem,” in *Symposium on Applied Computing,* TaiChung, Taiwan, 2011.

[87] R. Helgaon, R. Barr and J. Kennington, “Minimal spanning trees: An empirical investigation of parallel algorithms,” *Parallel Computing,* vol. 12, no. 1, p. 45–52, 1989.

[88] F. Dehne and S. Gotz, “Practical parallel algorithms for minimum spanning trees,” in *17th IEEE Symposium on Reliable Distributed Systems,* West Lafayette, IN, 1998.

[89] E. Gonina and L. Kale, “Parallel Prim’s algorithm on dense graphs with a novel extension,” 2007.

[90] R. Setia, A. Nedunchezhian and S. Balachandran, “A new parallel algorithm for minimum spanning tree problem,” in *16th IEEE International Conference on High Performance Computing,* 2009.

[91] V. Vineet, P. Harish, S. Patidar and P. Narayanan, “Fast minimum spanning trees for large graphs on the GPU,” in *Conference on High Performance Graphics 2009*, 2009.

[92] W. Wang, S. Guo, F. Yang and J. Chen, “GPU-Based Fast Minimum Spanning Tree Using Data Parallel Primitives,” in *Second International Conference on Information Engineering and Computer Science,* 2010.

[93] S. Rostrup, S. Srivastava and K. Singhal, “Fast and memory-efficient minimum spanning tree on the GPU,” *International Journal of Computational Science and Engineering,* vol. 8, no. 1, pp. 21-33, 2013.

[94] C. McCreesh and P. Prosser, “A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs,” in *International conference on principles and practice of constraint programming*, 2015.

[95] B. Archibald, F. Dunlop, R. Hoffmann, C. McCreesh, P. Prosser and J. Trimble, “Sequential and parallel solution-biased search for subgraph algorithms,” in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2019.

[96] S. Quer, A. Marcelli and G. Squillero, “The Maximum Common Subgraph Problem: A Parallel and Multi-Engine Approach,” *Computation,* vol. 8, p. 48, 2020.

[97] C. McCreesh and P. Prosser, “The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound,” *ACM Transactions on Parallel Computing (TOPC)*, vol. 2, p. 1–27, 2015.

[98] R. Hoffmann, C. Mccreesh, S. N. Ndiaye, P. Prosser, C. Reilly, C. Solnon and J. Trimble, “Observations from parallelising three maximum common (connected) subgraph algorithms,” in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2018.

[99] R. Kimmig, H. Meyerhenke and D. Strash, “Shared memory parallel subgraph enumeration,” in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017.

[100] P. P. Vuppuluri and P. Chellapilla, “Serial and parallel memetic algorithms for the bounded diameter minimum spanning tree problem,” *Expert Systems*, 2020.

[101] W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum and A. D. F. E. E. Lusk, Using MPI: portable parallel programming with the message-passing interface, vol. 1, MIT press, 1999.

Table 1: Real-world applications

S. No.	Problem	Real-world applications
1.	**Degree-Constrained Minimum Spanning Tree**	Communication systems, Computer networks, Design of integrated circuits, Energy networks and Transportation.
2.	**Bounded-Diameter Minimum Spanning Tree**	Operational Research (Network design) and Optical routing problem (optical WDM networks).
3.	**Minimum Label Spanning Tree**	Network design VLSI (Very-large scale integration), Geometric Optimization, Connected communications network and Multimodal transportation network design.
4.	**Leaf-Constrained Minimum Spanning Tree**	Location design, Circuit design and Network design.
5.	**Optimum Communication Spanning Tree**	Optimum communication networks, Design of telecommunication networks and Transportation networks.

Table 2: Degree-constrained minimum spanning tree (DCMST) problem

S. No.	Paper title	Author	Ref	Brief description
1.	A note on genetic algorithms for degree constrained spanning tree problems	Gengui Zhou and Mitsuo Gen	[4]	-GA optimization technique is used and compare results with Lower Bound using Greedy. -The problem sizes are 10-50. -GAs and the prüfer number are effective to deal with such spanning tree problems.
2.	A weighted coding in a genetic algorithm for the degree-constrained MST problem	Gunther R. Raidl and BA Julstrom	[81]	-Describes a novel coding of ST in a GA for the DCMST problem. -Based on prüfer's proof of Cayley's formula. -Contains nine complete graphs of 50 to 200 vertices. -Describes the K-MST problem and a Steady-State GA, encodes candidate solutions to the problem in a novel way.
3.	Skewed VNS enclosing second-order heuristics for the min-degree constrained minimum spanning tree problem	Pedro Martins and Mauricio C. De Souza	[36]	-VNS metaheuristics & VNS randomized methodology, enhanced version of second-order (ESO) repetitive technique and Kruskal based greedy heuristics are discussed and tested. -Problem size from 30 to 500 vertices.
4.	Ant colony optimization approaches to the degree-constrained minimum spanning tree problem	Yoon-teck Bau et al.	[8]	-Two approaches are used P-ACO and K-ACO -K-ACO performs better than a prüfer-coded evolutionary algorithm (F-EA). -Structured hard (SHRD) graphs are constructed using non-Euclidean distance. -Problem size from 15-30 vertices, degrees 2, 3, and 5.
5.	A hybrid Lagrangian particle swarm optimization algorithm for the degree-constrained minimum spanning tree problem	Andreas Ernst	[10]	-New hybrid heuristic combining particle swarm optimization, a Lagrangian heuristic are discussed and name as CoLaPSO -The shrd1000-1500 and the str1508-2008 use, 100-200 nodes. -Show better performance than previous heuristics for this problem.
6.	Degree constrained minimum spanning tree problem: a learning automata approach	Javad akbari torkestani	[12]	-Proposes a decentralized learning automata-based heuristic called LACT for approximating DCMST. -The size of problem instances is between 15 to 150 for shrd, 50 to 100 for crd, 50 to 200 for str, 50 to 70 for sym, and 20 to 1000 for random.

				-Shows an appropriate choice of learning rate which finds the optimal solution close enough to one.
7.	Comparison of algorithms for the degree constrained minimum spanning tree	Mohan Krishnamoorthy et al.	[13]	-Develop three heuristics simulated annealing (SA), GA-F, GA-P and a method based on problem space search (PSS). -Data sets are crd, sym, str, shrd with problem size from 30 to 100. -Develops a new combinatorial structure of the problem. -Conclude PSS works best over all the problems.
8.	The hybrid of depth-first search technique and Kruskal algorithm for solving the multiperiod degree constrained minimum spanning tree problem	Wamiliana et al.	[14]	- The hybrid depth-first search and Kruskal's discuss to solve MPDCMST -Problem size from 10 to 100 vertices. -Conclude when a vertex is connected in the previous period.
9.	A multiperiod degree constrained minimal spanning tree problem	Rakesh Kawatra	[15]	-Greedy-heuristics and branch and bound based on Lagrangian are used. -Problem size 40 to 100.
10.	Computational Aspects of the Greedy Algorithm for Solving the Multi-Period Degree Constrained Minimum Spanning Tree Problem	W. Wamiliana et al.	[18]	-WDA1, WDA2 & WDA are discussed. -600 Random Table Dataset
11.	The Design of Greedy Algorithm for Solving the Multi-Period Degree Constrained Minimum Spanning Tree Problem	W. Wamiliana et al.	[19]	-Design of a Simple Greedy Algorithm discussed. -Dataset contains order 10 to 250 & Weight Random from [1, 100]
12.	Euclidean bottleneck bounded-degree spanning tree ratios	A. Biniaz	[20]	-Hamiltonicity of the graph is discussed. -Based on the Hamiltonicity of the square of the biconnected graph.
13.	Euclidean bounded-degree spanning tree ratios	T. M. Chan	[24]	-Presenting the first improved upper bound. -Their Algorithm implement in linear time for any dimension *d*.
14.	A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem	GR Raidl, BA Julstrom	[81]	-Uses weighted coding, Prim's algorithm. -Problem size 50-200
15.	A network-flow technique for finding low-weight bounded-degree spanning trees	S. P. Fekete et al.	[26]	-A novel network-flow based algorithm is developed for Bounded-Degree ST.

16.	Degree-bounded minimum spanning trees	R. Jothi and B. Raghavachari	[25]	-Discuss on improved approximation analysis for Chan's degree-4 MST
17.	On improved bounds for bounded degree spanning trees for points in arbitrary dimension	Samuel Zbarsky	[29]	-Give the proof of upper bound by strong induction on the number of points -If spanning having no vertices of degree > 3, weight at most 1.559 times the weight of spanning tree.
18.	A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem	G.R. Raidl and BA Julstrom	[81]	-Identify the shortest spanning tree using Genetic codings of spanning trees -Problem size 50-200 vertices
19.	md-MST is NP-hard for d=3	A.M de Almeida et al.	[33]	-Introduced combinatorial problem which is NP-hard for d >= 3
20.	VNS and second order heuristics for the min-degree constrained minimum spanning tree problem	P. Martins and M.C. de Souza	[36]	-VNS & ESO discussed and developed greedy heuristic for md-MST -Problem size 30-500 vertices
21.	Min-degree constrained minimum spanning tree problem: New formulation via Miller-Tucker-Zemlin constraints	İ. Akgün and B C Tansel	[34]	-Proposed degree-enforcing constraints and used Miller-Tucker-Zemlin sub-tour elimination constraints
22.	The min-degree constrained minimum spanning tree problem: Formulations and Branch-and-cut algorithm	L.C. Martinez and A.S. Da Cunha	[35]	-Branch-and-cut algorithm is used -Problem size 30-100 vertices

Table 3: Bounded-Diameter Minimal Spanning Tree (BDMST) Problem

S. No.	Paper title	Author	Ref	Brief description
1.	Random-tree diameter and the diameter constrained minimum spanning tree	Ayman Abdalla et al.	[41]	-Investigate the behavior of the diameter of MST in a Randomly weighted complete graph -OTTC, IR1, IR2 discussed and conclude IR2 faster than OTTC
2.	Network flow models for designing diameter constrained minimum spanning and Steiner trees	Luis Gouveia and Thomas L. Magnanti	[40]	-Examine network flow-based -Problem size 20-100 vertices & edges 100-800
3.	Greedy heuristics for the bounded diameter minimum spanning tree problem	BA Julstrom	[43]	-A Greedy Heuristic & A Randomized algorithm is discussed and Compare results -Problem 100-1000 vertices

4.	Improved heuristics for the bounded-diameter minimum spanning tree problem	Alok Singh and AK Gupta	[47]	-Randomized greedy heuristic & center-based tree construction discussed -Problem size 50-1000
5.	New multiparent recombination in genetic algorithm for solving bounded diameter minimum spanning tree problem	Huynh Binh & Nguyen Duc Nghia	[49]	-Proposed 3 multi-parent recombination operators -OTTC, RGH, RGH1 & RGH2 are described

Table 4: Minimum Label Spanning Tree (MLST) Problem

S. No.	Paper title	Author	Ref	Brief description
1.	Metaheuristics comparison for the minimum labeling spanning tree problem	R. Cerulli et al.	[62]	-Discussed & apply k-switch neighborhood, Bit-flip neighborhood. -Problem size 50-100
2.	A one-parameter genetic algorithm for the minimum labeling spanning tree problem	Y. Xiong et al.	[63]	-Use one-parameter GA & MVCA. -Problem size 20-200
3.	An effective genetic algorithm for the minimum label spanning tree problem	Jeremiah Nummela et al.	[65]	-GA & a greedy heuristic is used. -Problem size 200-400 vertices & 1000-2000 edges.
4.	Solving the Minimum Label Spanning Tree Problem by Ant Colony Optimization	A.M. Chwatal	[66]	-Present application of Ant Colony Optimization -Problem size 200-500

Table 5: Leaf-Constrained Minimum Spanning Tree Problem

S. No.	Paper title	Author	Ref	Brief description
1.	Codings and Operators in two genetic algorithms for the Leaf-Constrained Minimum Spanning Tree Problem	Julstrom	[69]	-Discuss Blob-coded Ga & Subset-coded GA. -Problem size 50-250 vertices.
2.	New Metaheuristic Approaches for the Leaf-Constrained Minimum Spanning Tree Problem	Alok Singh and Anurag Singh Baghel	[71]	-Two algorithms discussed to solve LCMSTP 1. Ant Colony Optimization (ACO) 2. 65 test problem instances on 50-300 vertices

3.	An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem	Alok Singh	[72]	-Propose Ant Bee Colony (ABC) Algorithm -Instances 1-15 & 50-300 nodes -Inspired by the behavior of honey Bee Swarm
4.	Leaf constrained minimal spanning trees solved by modified quantum-behaved particle swarm optimization	Saeed Farzi and Ahmad Barzani	[73]	-Propose an approach based on Quantum-Behaved Particle Swarm Optimization (QPSO) Problem -Problem size 50-300 vertices -By combining classical PSO and Quantum mechanics

Table 6: Optimum Communicating Spanning Tree Problem

S. No.	Paper title	Author	Ref	Brief description
1.	Optimum Communication Spanning Trees	T.C. Hu	[75]	-Deal with two special cases other than general communication -Test on the converted graph with 6 nodes
2.	Exact and Heuristic Algorithms for the Optimum Communication Spanning Tree Problem	R.K. Ahuja and VVS Murthy	[77]	-Develop an algorithm based on branch-and-bound and a heuristic -Test on 40 nodes & 69 arcs
3.	A Novel Particle Swarm Optimization Based Algorithm for the Optimal Communication Spanning Tree Problem	Anh Taun Hoang et al.	[85]	-Develop an algorithm based on PSO & Node Biased Encoding (NBE) Scheme -Problem size 10-50
4.	New Hybrid Genetic Algorithm for Solving Optimal Communication Spanning Tree Problem	Pham Trung Kien et al.	[86]	-Describe Hybrid Genetic Algorithm & Combine with Particle Swarm Optimization (PSO) -Problem size of 200 nodes